

## II. Working with data in R (presentation)

Data Science Lab, University of Copenhagen

2026-05-06

### Table of contents

Tidyverse package . . . . .	1
Import data, inspect data . . . . .	2
About the data . . . . .	4
Extracting variables, simple summary statistics . . . . .	4
Filter data (selecting rows): <code>filter</code> . . . . .	6
Select variables: <code>select</code> . . . . .	8
Transformations of data: <code>mutate</code> . . . . .	9
Counting, tabulation of categorical variables: <code>count</code> . . . . .	11
Sort data: <code>arrange</code> . . . . .	12
Group data: <code>group_by</code> . . . . .	14
Summary statistics, revisited: <code>summarize</code> . . . . .	15
Merge datasets with different columns: <code>join</code> . . . . .	18
Stack datasets with the same variables: <code>bind_rows</code> . . . . .	19
The pipe operator: <code>%&gt;%</code> . . . . .	20

### Tidyverse package

The tidyverse is a collection of R packages which, among other things, facilitate data handling and data transformation in R. See <https://www.tidyverse.org/> for details.

We must install and load the R package **tidyverse** before we have access to the functions.

- Install package: One option is to go via the *Tools* menu: *Tools* → *Install packages* → write **tidyverse** in the field called *Packages*. This only has to be done once.
- Load package: Use the `library` command below (preferred), or go to the *Packages* menu in the bottom right window, find **tidyverse** in the list, and click it. This has to be done in every R-session where you use the package.

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr      1.2.0      v readr      2.1.6
v forcats    1.0.1      v stringr    1.6.0
v ggplot2    4.0.2      v tibble     3.3.1
v lubridate  1.9.5      v tidyr      1.3.2
v purrr      1.2.1
-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()     masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become
```

We can see that loading the **tidyverse** package here results in some technical messages from R. You do not have to worry about this (even if some of the text is coloured red).

---

## Import data, inspect data

Data from Excel files can be imported via the *Import Dataset* facility. You may get the message that the package **readxl** should be installed. If so, then install it as explained for **tidyverse** above.

- Find *Import Data* in the upper right window in RStudio, and choose *From Excel* in the dropdown menu.
- A new window opens. Browse for the relevant Excel file; then a preview of the dataset is shown. Check that it looks OK, and click *Import*.
- Three things happened: Three lines of code was generated (and executed) in the Console, a new dataset now appears in the Environment window, and the dataset is shown in the top left window. Check again that it looks OK.
- Copy the first two lines of code into your R script (or into an R chunk in your Markdown document), but delete line starting with **View** and write instead the name of the dataset, here **downloads**. Then the first 10 lines of the data set are printed.

```
library(readxl)
downloads <- read_excel("downloads.xlsx")
downloads
```

```
# A tibble: 147,035 x 6
  machineName userID size time date month
  <chr>      <dbl> <dbl> <dbl> <dtm> <chr>
1 cs18      146579 2464 0.493 1995-04-24 00:00:00 1995-04
2 cs18      995988 7745 0.326 1995-04-24 00:00:00 1995-04
3 cs18      317649 6727 0.314 1995-04-24 00:00:00 1995-04
4 cs18      748501 13049 0.583 1995-04-24 00:00:00 1995-04
5 cs18      955815 356 0.259 1995-04-24 00:00:00 1995-04
6 cs18      444174 0 0 1995-04-24 00:00:00 1995-04
7 cs18      446911 0 0 1995-04-24 00:00:00 1995-04
8 cs18      449552 0 0 1995-04-24 00:00:00 1995-04
9 cs18      456142 0 0 1995-04-24 00:00:00 1995-04
10 cs18      458942 0 0 1995-04-24 00:00:00 1995-04
# i 147,025 more rows
```

R has stored the data in a so-called *tibble*, a type of data frame. Rows are referred to as *observations* or *data lines*, columns as *variables*. The data rows appear in the order as in the Excel file.

Notice the line with data types below the variable names. `to` character variables, `to` data-and-time variables, `(double)` refer to numerical variables. It is often useful to convert variables of type `to` factor variables, as illustrated later in the document.

It is often useful to get a summary of each of the variables in the dataset:

```
summary(downloads)
```

```
machineName      userID      size      time
Length:147035    Min.   :    0    Min.   :    0    Min.   : 0.000
Class :character 1st Qu.:256435 1st Qu.:    0    1st Qu.: 0.000
Mode  :character Median :507567 Median :    0    Median : 0.000
              Mean  :504923 Mean  :  4154    Mean  : 0.954
              3rd Qu.:753920 3rd Qu.:    0    3rd Qu.: 0.000
              Max.   :999989 Max.   :14518894 Max.   :1878.076

      date      month
Min.   :1994-11-22 00:00:00 Length:147035
1st Qu.:1995-02-07 00:00:00 Class :character
Median :1995-02-23 00:00:00 Mode  :character
Mean   :1995-03-04 22:17:15
3rd Qu.:1995-04-12 00:00:00
Max.   :1995-05-18 00:00:00
```

At this point, the output is mainly useful for the numerical variables (and date variables). We will later see how categorical variables can be converted to so-called factors, and that the output is also useful for such variables.

**Digression:** If data are saved in a csv file (comma separated values), possibly generated via an Excel sheet, then data can be read with the `read.csv` function. For example, if the data file is called `mydata.csv` and values are separated with commas, then the command

```
mydata <- read.csv("mydata.csv", sep=",")
```

creates a data frame in R with the data. The data frame is *not* a tibble and some of the commands below would not work for such a data frame. Alternatively, you can also import csv files via the *From Text* item in the *Import Dataset* menu.

---

## About the data

The dataset is from Boston University and is about www data transfers from November 1994 to May 1995.

- It has 147,035 data lines and 6 variables
- We will use 3 of the variables: *machineName* is the name of the server from which the file was downloaded, *size* is the download size in bytes, and *time* is the download time in seconds.

---

## Extracting variables, simple summary statistics

Variables can be extracted with the `$`-syntax, and we can use squared brackets to show only the first 40, say, values.

```
time_vector <- downloads$time  
time_vector[1:40]
```

```
[1] 0.493030 0.325608 0.313704 0.582537 0.259252 0.000000 0.000000 0.000000  
[9] 0.000000 0.000000 0.000000 0.335502 0.284853 0.000000 0.000000 0.000000  
[17] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000  
[25] 0.285665 0.397111 3.410561 0.267474 0.842364 0.903005 2.784645 2.806157  
[33] 0.990092 0.477629 0.000000 0.000000 0.000000 0.000000 0.988944 0.000000
```

Summary statistics like mean, standard deviation, median are easily computed for a vector.

Examples of R functions for computing summary statistics: `length`, `mean`, `median`, `sd`, `var`, `sum`, `quantile`, `min`, `max`, `IQR`.

```
length(time_vector)
```

```
[1] 147035
```

```
mean(time_vector)
```

```
[1] 0.9539674
```

```
sd(time_vector)
```

```
[1] 14.22557
```

```
median(time_vector)
```

```
[1] 0
```

```
min(time_vector)
```

```
[1] 0
```

Notice that more than half the observations have time equal to zero (median is zero).

With the `summary` function, many of the above summary statistics (but not the standard deviation) are computed in one go:

```
summary(time_vector)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.000	0.000	0.000	0.954	0.000	1878.076

---

## Filter data (selecting rows): filter

The `filter` function is used to make sub-datasets where only certain datalines (rows) are maintained. It is described with *logical expressions* which datalines should be kept in the dataset.

Say that we only want observations with download time larger than 1000 seconds; there happens to be eight such observations:

```
# Note: Here we use the filter() function from the tidyverse-package.
#       If the tidyverse-package was not loaded, then we would use
#       the filter() function from the stats-package. And actually
#       get an error message.
filter(downloads, time > 1000)
```

```
# A tibble: 8 x 6
  machineName userID    size time date      month
  <chr>      <dbl>   <dbl> <dbl> <dtm>    <chr>
1 cs18      502807 4055821 1275. 1994-12-02 00:00:00 1994-12
2 cs18      16653 2573336 1335. 1994-11-22 00:00:00 1994-11
3 cs18      957883 2743516 1151. 1994-11-22 00:00:00 1994-11
4 cs18      47910 4720220 1749. 1994-11-22 00:00:00 1994-11
5 tweetie   223655  245003 1214. 1995-04-13 00:00:00 1995-04
6 kermit    576790 14518894 1380. 1995-04-20 00:00:00 1995-04
7 kermit    139654 1079731 1129. 1995-02-23 00:00:00 1995-02
8 pluto     337530 8674562 1878. 1995-03-13 00:00:00 1995-03
```

Or say that only want observations with strictly positive download size:

```
downloads2 <- filter(downloads, size > 0)
downloads2
```

```
# A tibble: 36,708 x 6
  machineName userID    size time date      month
  <chr>      <dbl>   <dbl> <dbl> <dtm>    <chr>
1 cs18      146579  2464 0.493 1995-04-24 00:00:00 1995-04
2 cs18      995988  7745 0.326 1995-04-24 00:00:00 1995-04
3 cs18      317649  6727 0.314 1995-04-24 00:00:00 1995-04
4 cs18      748501 13049 0.583 1995-04-24 00:00:00 1995-04
5 cs18      955815   356 0.259 1995-04-24 00:00:00 1995-04
6 cs18      596819 15063 0.336 1995-04-24 00:00:00 1995-04
7 cs18      169424  2548 0.285 1995-04-24 00:00:00 1995-04
```

```

8 cs18          386686  1932 0.286 1995-04-24 00:00:00 1995-04
9 cs18          783767  7294 0.397 1995-04-24 00:00:00 1995-04
10 cs18         788633  4470 3.41  1995-04-24 00:00:00 1995-04
# i 36,698 more rows

```

Notice that this result is assigned to **downloads2**. It has 36,708 data lines. The original data called **downloads** still exists with 147,035 data lines.

Filtering requires *logical predicates*. These are expressions in terms of columns, which evaluate to either TRUE or FALSE for each row. Logical expressions can be combined with logical operations.

- Comparisons: ==, !=, <, >, <=, >=, %in%, is.na
- Logical operations: ! (not), | (or), & (and). A comma can be used instead of '&'

Here comes two sub-datasets:

```

# Rows from kermit, and with size greater than 200000 bytes are kept.
filter(downloads2, machineName == "kermit", size > 200000)

```

```

# A tibble: 98 x 6
  machineName userID      size      time date      month
  <chr>      <dbl>    <dbl>    <dbl> <dtm>      <chr>
1 kermit    157161  498325  0.629 1995-04-13 00:00:00 1995-04
2 kermit    734988  271058  17.3   1995-04-22 00:00:00 1995-04
3 kermit    388066  435923  29.2   1995-04-22 00:00:00 1995-04
4 kermit     34030  642771  4.80   1995-04-12 00:00:00 1995-04
5 kermit    327021  724757  4.98   1995-04-12 00:00:00 1995-04
6 kermit     38016  561762  9.75   1995-04-05 00:00:00 1995-04
7 kermit    277395  404209  11.3   1995-04-05 00:00:00 1995-04
8 kermit    576790 14518894 1380.   1995-04-20 00:00:00 1995-04
9 kermit     17623  489473  21.2   1995-04-20 00:00:00 1995-04
10 kermit   198041  355963  15.3   1995-04-20 00:00:00 1995-04
# i 88 more rows

```

```

# Rows NOT from kermit, and with size greater than 200000 bytes are kept.
filter(downloads2, machineName != "kermit", size > 200000)

```

```

# A tibble: 220 x 6
  machineName userID      size      time date      month
  <chr>      <dbl>    <dbl>    <dbl> <dtm>      <chr>
1 cs18      204764 2691689  0.834 1995-04-26 00:00:00 1995-04

```

```

2 cs18      397405  215045    1.10  1994-12-15 00:00:00 1994-12
3 cs18      809091  226586    3.92  1994-12-15 00:00:00 1994-12
4 cs18      779032 1080472   156.   1994-12-11 00:00:00 1994-12
5 cs18      688294  748705    93.1   1994-12-11 00:00:00 1994-12
6 cs18      447740 6360764   863.   1994-12-11 00:00:00 1994-12
7 cs18      708452  204918    7.07  1994-12-18 00:00:00 1994-12
8 cs18      598668  204918   12.7   1994-12-18 00:00:00 1994-12
9 cs18      288167  204918    4.98  1994-12-18 00:00:00 1994-12
10 cs18     974956  203714    6.13  1994-12-16 00:00:00 1994-12
# i 210 more rows

```

---

### Select variables: `select`

Sometimes, datasets has many variables of which only some are relevant for the analysis. Variables can be selected or skipped with the `select` function.

```

# Without the date variable
select(downloads2, -date)

```

```

# A tibble: 36,708 x 5
  machineName userID  size  time month
  <chr>         <dbl> <dbl> <dbl> <chr>
1 cs18      146579  2464 0.493 1995-04
2 cs18      995988  7745 0.326 1995-04
3 cs18      317649  6727 0.314 1995-04
4 cs18      748501 13049 0.583 1995-04
5 cs18      955815   356 0.259 1995-04
6 cs18      596819 15063 0.336 1995-04
7 cs18      169424  2548 0.285 1995-04
8 cs18      386686  1932 0.286 1995-04
9 cs18      783767  7294 0.397 1995-04
10 cs18      788633  4470 3.41  1995-04
# i 36,698 more rows

```

```

# Only include the three mentioned variable names
downloads3 <- select(downloads2, machineName, size, time)
downloads3

```



```
# A tibble: 36,708 x 3
  machineName size time
  <chr>      <dbl> <dbl>
1 cs18        2464 0.493
2 cs18        7745 0.326
3 cs18        6727 0.314
4 cs18       13049 0.583
5 cs18         356 0.259
6 cs18       15063 0.336
7 cs18        2548 0.285
8 cs18        1932 0.286
9 cs18        7294 0.397
10 cs18       4470 3.41
# i 36,698 more rows
```

Notice that we have made a new dataframe, **downloads3** with only three variables.

---

## Transformations of data: mutate

Transformations of existing variables in the data set can be computed and included in the data set with the `mutate` function.

We first compute two new variables, download speed (**speed**) and the logarithm of the download size (**logSize**):

```
# Two new variables
downloads3 <- mutate(downloads3, speed = size / time, logSize = log10(size))
downloads3
```

```
# A tibble: 36,708 x 5
  machineName size time speed logSize
  <chr>      <dbl> <dbl> <dbl> <dbl>
1 cs18        2464 0.493 4998.    3.39
2 cs18        7745 0.326 23786.   3.89
3 cs18        6727 0.314 21444.   3.83
4 cs18       13049 0.583 22400.   4.12
5 cs18         356 0.259 1373.    2.55
6 cs18       15063 0.336 44897.   4.18
7 cs18        2548 0.285 8945.    3.41
```

```

 8 cs18          1932 0.286  6763.    3.29
 9 cs18          7294 0.397 18368.    3.86
10 cs18          4470 3.41   1311.    3.65
# i 36,698 more rows

```

We then make a new categorical variable, **speedCat**, which is “Slow” is speed < 150 and “Fast” otherwise:

```

# New variable with if-else construction
downloads3 <- mutate(downloads3, speedCat = ifelse(speed < 150, "Slow", "Fast"))
downloads3

```

```

# A tibble: 36,708 x 6
  machineName size time speed logSize speedCat
  <chr>      <dbl> <dbl> <dbl>   <dbl> <chr>
1 cs18        2464 0.493  4998.    3.39 Fast
2 cs18        7745 0.326 23786.    3.89 Fast
3 cs18        6727 0.314 21444.    3.83 Fast
4 cs18       13049 0.583 22400.    4.12 Fast
5 cs18         356 0.259  1373.    2.55 Fast
6 cs18       15063 0.336 44897.    4.18 Fast
7 cs18        2548 0.285  8945.    3.41 Fast
8 cs18        1932 0.286  6763.    3.29 Fast
9 cs18        7294 0.397 18368.    3.86 Fast
10 cs18        4470 3.41   1311.    3.65 Fast
# i 36,698 more rows

```

Categorical variables are coded as “character variables” when data are imported, but it is often preferable to code them as so-called factors instead. We can use the function **factor** in combination with **mutate** to obtain this. Here, we overwrite the old version of **machineName** with the new factor version. Notice how the output from **summary** now shown the number of observations for each machine.

```

# Change machineName to factor
downloads3 <- mutate(downloads3, machineName=factor(machineName))
summary(downloads3)

```

machineName	size	time	speed
cs18 : 3814	Min. : 3	Min. :4.369e-02	Min. : 0.3
kermit : 9094	1st Qu.: 953	1st Qu.:4.426e-01	1st Qu.: 1060.5
piglet :11200	Median : 2188	Median :7.314e-01	Median : 2902.8

pluto	: 5253	Mean	: 16638	Mean	: 3.820e+00	Mean	: 9990.4
tweetie	: 7347	3rd Qu.:	6500	3rd Qu.:	1.893e+00	3rd Qu.:	7126.4
		Max.	:14518894	Max.	:1.878e+03	Max.	:3228695.3

  

logSize	speedCat		
Min.	:0.4771	Length:	36708
1st Qu.:	2.9791	Class	:character
Median	:3.3400	Mode	:character
Mean	:3.4475		
3rd Qu.:	3.8129		
Max.	:7.1619		

---

## Counting, tabulation of categorical variables: count

The `count` function is useful for counting datalines, possibly according to certain criteria or for the different levels of categorical values.

```
# Total number of observations in the current dataset
count(downloads3)
```

```
# A tibble: 1 x 1
      n
  <int>
1 36708
```

```
# Number of observations from each machine
count(downloads3, machineName)
```

```
# A tibble: 5 x 2
  machineName      n
  <fct>         <int>
1 cs18          3814
2 kermit         9094
3 piglet        11200
4 pluto          5253
5 tweetie        7347
```

```
# Number of observations which have/have not size larger than 5000
count(downloads3, size>5000)
```

```
# A tibble: 2 x 2
  `size > 5000`      n
    <lgl>          <int>
1 FALSE          25865
2 TRUE           10843

# Number of observations for each combination of machine name and the *slow* variable.
count(downloads3, machineName, speedCat)
```

```
# A tibble: 10 x 3
  machineName speedCat      n
    <fct>      <chr>    <int>
1 cs18       Fast      3662
2 cs18       Slow       152
3 kermit     Fast      8717
4 kermit     Slow       377
5 piglet     Fast     10734
6 piglet     Slow       466
7 pluto      Fast      4963
8 pluto      Slow       290
9 tweetie    Fast      6983
10 tweetie    Slow       364
```

---

## Sort data: arrange

The `arrange` function can be used to sort the data according to one or more columns.

Let us sort the data according to download time (ascending order). The first lines of the sorted data set is printed on-screen, but the dataset **downloads3** has *not* been changed.

```
arrange(downloads3, time)
```

```
# A tibble: 36,708 x 6
  machineName  size  time  speed logSize speedCat
    <fct>      <dbl> <dbl>   <dbl>   <dbl> <chr>
1 pluto        1806 0.0437  41335.    3.26 Fast
2 tweetie      7747 0.0462 167644.    3.89 Fast
3 piglet        353 0.0525   6729.    2.55 Fast
4 piglet     18110 0.0535 338359.    4.26 Fast
```

```

5 kermit      454 0.0550  8248.    2.66 Fast
6 tweetie     452 0.0551  8208.    2.66 Fast
7 tweetie     452 0.0554  8155.    2.66 Fast
8 tweetie     452 0.0564  8010.    2.66 Fast
9 tweetie     452 0.0566  7991.    2.66 Fast
10 kermit      452 0.0602  7514.    2.66 Fast
# i 36,698 more rows

```

Two different examples:

```

# According to download size in descending order
arrange(downloads3, desc(time))

```

```

# A tibble: 36,708 x 6
  machineName      size  time  speed logSize speedCat
  <fct>          <dbl> <dbl> <dbl>   <dbl> <chr>
1 pluto         8674562 1878.  4619.    6.94 Fast
2 cs18          4720220 1749.  2700.    6.67 Fast
3 kermit       14518894 1380. 10522.    7.16 Fast
4 cs18          2573336 1335.  1928.    6.41 Fast
5 cs18          4055821 1275.  3180.    6.61 Fast
6 tweetie       245003 1214.   202.    5.39 Fast
7 cs18          2743516 1151.  2383.    6.44 Fast
8 kermit        1079731 1129.   956.    6.03 Fast
9 cs18          6360764  863.  7374.    6.80 Fast
10 cs18         1679580  755.  2226.    6.23 Fast
# i 36,698 more rows

```

```

# After machine name and then according to download size in descending order
arrange(downloads3, machineName, desc(time))

```

```

# A tibble: 36,708 x 6
  machineName      size  time  speed logSize speedCat
  <fct>          <dbl> <dbl> <dbl>   <dbl> <chr>
1 cs18          4720220 1749.  2700.    6.67 Fast
2 cs18          2573336 1335.  1928.    6.41 Fast
3 cs18          4055821 1275.  3180.    6.61 Fast
4 cs18          2743516 1151.  2383.    6.44 Fast
5 cs18          6360764  863.  7374.    6.80 Fast
6 cs18          1679580  755.  2226.    6.23 Fast
7 cs18          1736924  744.  2335.    6.24 Fast

```

```

8 cs18          1683412  695. 2423.    6.23 Fast
9 cs18          1679580  684. 2457.    6.23 Fast
10 cs18         1273543  633. 2011.    6.11 Fast
# i 36,698 more rows

```

---

## Group data: `group_by`

We can group the dataset by one or more categorical variables with `group_by`. The dataset is not changed as such, but - as we will see - grouping can be useful for computation of summary statistics and graphics.

Here we group after machine name (first) and after machine name *and* the categorical speed variable (second). The only way we can see it at this point is in the second line in the output (`# Groups:`):

```

# Group according to machine
group_by(downloads3, machineName)

```

```

# A tibble: 36,708 x 6
# Groups:   machineName [5]
  machineName size time speed logSize speedCat
  <fct>      <dbl> <dbl> <dbl> <dbl> <chr>
1 cs18        2464 0.493 4998. 3.39 Fast
2 cs18        7745 0.326 23786. 3.89 Fast
3 cs18        6727 0.314 21444. 3.83 Fast
4 cs18       13049 0.583 22400. 4.12 Fast
5 cs18         356 0.259 1373. 2.55 Fast
6 cs18       15063 0.336 44897. 4.18 Fast
7 cs18        2548 0.285 8945. 3.41 Fast
8 cs18        1932 0.286 6763. 3.29 Fast
9 cs18        7294 0.397 18368. 3.86 Fast
10 cs18       4470 3.41 1311. 3.65 Fast
# i 36,698 more rows

```

```

# Group according to machine and slow
group_by(downloads3, machineName, speedCat)

```

```
# A tibble: 36,708 x 6
# Groups:   machineName, speedCat [10]
  machineName size time speed logSize speedCat
  <fct>      <dbl> <dbl> <dbl> <dbl> <chr>
1 cs18        2464 0.493 4998.    3.39 Fast
2 cs18        7745 0.326 23786.   3.89 Fast
3 cs18        6727 0.314 21444.   3.83 Fast
4 cs18       13049 0.583 22400.   4.12 Fast
5 cs18         356 0.259 1373.    2.55 Fast
6 cs18       15063 0.336 44897.   4.18 Fast
7 cs18        2548 0.285 8945.    3.41 Fast
8 cs18        1932 0.286 6763.    3.29 Fast
9 cs18        7294 0.397 18368.   3.86 Fast
10 cs18       4470 3.41 1311.    3.65 Fast
# i 36,698 more rows
```

---

## Summary statistics, revisited: `summarize`

Recall how we could compute summary statistics for a single variable in a dataset, e.g.

```
mean(downloads3$size)
```

```
[1] 16638.36
```

```
max(downloads3$size)
```

```
[1] 14518894
```

With `summarize` we can compute summary statistics for a variable for each level of a grouping variable or for each combination of several grouping variables.

First, a bunch of summaries for the size variable for each machine name, where we give explicit names for the new variables:

```
downloads.grp1 <- group_by(downloads3, machineName)
summarize(downloads.grp1,
  avg = mean(size),
  med = median(size),
  stdev = sd(size),
  total = sum(size),
  n = n())
```

```
# A tibble: 5 x 6
  machineName avg med stdev total n
  <fct>      <dbl> <dbl> <dbl> <dbl> <int>
1 cs18      26375. 1990. 208915. 100593281 3814
2 kermit    19247. 2466  213985. 175032552 9094
3 piglet    14121. 2146. 188340. 158149841 11200
4 pluto     13822. 2069  144425.  72605544 5253
5 tweetie   14207. 2197   94318. 104379794 7347
```

Second, the same thing but for each combination of machine name and the categorical speed variable:

```
downloads.grp2 <- group_by(downloads3, machineName, speedCat)
summarize(downloads.grp2,
  avg = mean(size),
  med = median(size),
  stdev = sd(size),
  total = sum(size),
  n = n())
```

```
`summarise()` has regrouped the output.
i Summaries were computed grouped by machineName and speedCat.
i Output is grouped by machineName.
i Use `summarise(.groups = "drop_last")` to silence this message.
i Use `summarise(.by = c(machineName, speedCat))` for per-operation grouping
  (`?dplyr::dplyr_by`) instead.
```

```
# A tibble: 10 x 7
# Groups:   machineName [5]
  machineName speedCat avg med stdev total n
  <fct>      <chr>   <dbl> <dbl> <dbl> <dbl> <int>
1 cs18      Fast    27445. 2092. 213140. 100503042 3662
```



2	cs18	Slow	594.	368.	614.	90239	152
3	kermit	Fast	20030.	2598	218529.	174602282	8717
4	kermit	Slow	1141.	541	3049.	430270	377
5	piglet	Fast	14687.	2264	192365.	157650747	10734
6	piglet	Slow	1071.	416.	1934.	499094	466
7	pluto	Fast	14564.	2164	148551.	72280790	4963
8	pluto	Slow	1120.	413	2108.	324754	290
9	tweetie	Fast	14894.	2373	96694.	104001733	6983
10	tweetie	Slow	1039.	471	2603.	378061	364

Third, mean and standard deviation for several variables:

```
summarize(downloads.grp2, across(c("time", "size"), list(ave=mean, stdev=sd)))
```

```
`summarise()` has regrouped the output.
i Summaries were computed grouped by machineName and speedCat.
i Output is grouped by machineName.
i Use `summarise(.groups = "drop_last")` to silence this message.
i Use `summarise(.by = c(machineName, speedCat))` for per-operation grouping
  (`?dplyr::dplyr_by`) instead.
```

```
# A tibble: 10 x 6
# Groups:   machineName [5]
  machineName speedCat time_ave time_stdev size_ave size_stdev
  <fct>        <chr>    <dbl>    <dbl>    <dbl>    <dbl>
1 cs18        Fast      5.17     57.1    27445.    213140.
2 cs18        Slow      9.63     17.8     594.      614.
3 kermit       Fast      3.41     25.3    20030.    218529.
4 kermit       Slow     20.7     47.8     1141.     3049.
5 piglet       Fast      2.33     13.8    14687.    192365.
6 piglet       Slow     19.4     40.2     1071.     1934.
7 pluto        Fast      3.40     30.4    14564.    148551.
8 pluto        Slow     21.7     46.3     1120.     2108.
9 tweetie      Fast      2.68     17.3    14894.     96694.
10 tweetie     Slow     17.8     34.5     1039.     2603.
```

The datasets with summaries can be saved as datasets themselves, for example to be used as the basis for certain graphs.

## Merge datasets with different columns: join

Consider the situation where data from the same “individuals” are spread in different datasets, and we want to combine the data into one dataset. This is possible with the “join” functions which are available with the **tidyverse** package. There are several variants; here we just show how to use the one called *full\_join* in a tiny example. Say we want to merge the dataset consisting of the first five datalines from **downloads3** with another dataset which has six observations. They are linked together via a variable called **id** and we want a dataset which merges the two datasets, keeping observations with the same id together.

```
# Subset of the downloads3 data
downloads4 <- downloads3[1:5,]
# Make an id variable
downloads4$id <- 1:5
downloads4
```

```
# A tibble: 5 x 7
  machineName size  time  speed logSize speedCat   id
  <fct>      <dbl> <dbl>  <dbl>  <dbl> <chr>   <int>
1 cs18        2464 0.493  4998.    3.39 Fast     1
2 cs18        7745 0.326 23786.    3.89 Fast     2
3 cs18        6727 0.314 21444.    3.83 Fast     3
4 cs18       13049 0.583 22400.    4.12 Fast     4
5 cs18         356 0.259  1373.    2.55 Fast     5
```

```
# The "extra" dataset. It has two variables: id and x
extraData <- tibble(id=6:1, x = c(16,15,14,13,12,11))
extraData
```

```
# A tibble: 6 x 2
   id     x
  <int> <dbl>
1     6    16
2     5    15
3     4    14
4     3    13
5     2    12
6     1    11
```

```
# Merging
full_join(downloads4, extraData)
```

Joining with ``by = join_by(id)``

```
# A tibble: 6 x 8
  machineName size   time speed logSize speedCat   id     x
  <fct>       <dbl> <dbl> <dbl>   <dbl> <chr>   <int> <dbl>
1 cs18         2464 0.493 4998.    3.39 Fast     1     11
2 cs18         7745 0.326 23786.   3.89 Fast     2     12
3 cs18         6727 0.314 21444.   3.83 Fast     3     13
4 cs18        13049 0.583 22400.   4.12 Fast     4     14
5 cs18          356 0.259 1373.    2.55 Fast     5     15
6 <NA>          NA  NA      NA      NA    <NA>     6     16
```

The merged dataset has six datalines, but the last line has NA's for all variables except **ID** and **x** because no observation with `id=6` exists in **downloads4**. One can use different join functions to leave out this dataline.

### Stack datasets with the same variables: `bind_rows`

Consider another situation where we have two datasets with the same variables, and want to stack them together. The function `bind_rows` is useful for that. Here, for illustration, we first make two small subsets of the data, and then combine them.

```
# Make small datasets and combine them
part1 <- downloads3[1:3,]
part2 <- downloads3[10001:10003,]
part1
```

```
# A tibble: 3 x 6
  machineName size   time speed logSize speedCat
  <fct>       <dbl> <dbl> <dbl>   <dbl> <chr>
1 cs18         2464 0.493 4998.    3.39 Fast
2 cs18         7745 0.326 23786.   3.89 Fast
3 cs18         6727 0.314 21444.   3.83 Fast
```

```
part2
```

```
# A tibble: 3 x 6
  machineName size   time speed logSize speedCat
  <fct>       <dbl> <dbl> <dbl>   <dbl> <chr>
1 kermit       2747 0.541 5074.    3.44 Fast
2 kermit       2150 0.404 5327.    3.33 Fast
3 kermit       7998 0.748 10688.   3.90 Fast
```

```
bind_rows(part1, part2)
```

```
# A tibble: 6 x 6
  machineName size time speed logSize speedCat
  <fct>      <dbl> <dbl> <dbl> <dbl> <chr>
1 cs18      2464 0.493 4998. 3.39 Fast
2 cs18      7745 0.326 23786. 3.89 Fast
3 cs18      6727 0.314 21444. 3.83 Fast
4 kermit    2747 0.541 5074. 3.44 Fast
5 kermit    2150 0.404 5327. 3.33 Fast
6 kermit    7998 0.748 10688. 3.90 Fast
```

---

## The pipe operator: %>%

Two or more function calls can be evaluated sequentially using the so-called pipe operator, %>%. Nesting of function calls becomes more readable, and intermediate assignments are avoided.

Let us try it to do a bunch of things in one go, starting with the original dataset. Notice also the extended use of summarize.

```
downloads %>%
  filter(size>0) %>% # Subset of data
  group_by(machineName) %>% # Grouping
  summarize(across(c("time","size"),list(avg=mean, sd=sd))) %>% # Compute all means and sd's
  arrange(size_avg) # Sort after mean
```

```
# A tibble: 5 x 5
  machineName time_avg time_sd size_avg size_sd
  <chr>      <dbl> <dbl> <dbl> <dbl>
1 pluto      4.41 31.8 13822. 144425.
2 piglet     3.04 16.1 14121. 188340.
3 tweetie    3.43 18.8 14207. 94318.
4 kermit     4.12 26.8 19247. 213985.
5 cs18       5.35 56.1 26375. 208915.
```

End of presentation.